CHAIR OF DECENTRALIZED INFORMATION SYSTEMS & DATA MANAGEMENT

TECHNICAL UNIVERSITY OF MUNICH

Thesis proposal

Concurrent Range Lock

Author: **Thua Duc Nguyen**

Supervisor: **Prof. Dr. Viktor Leis**

Advisor: **Lam Duy Nguyen**

1 Introduction

Range locks acquire exclusive locks on consecutive values within a specified range. They play a vital role in filesystems, operating systems, and databases[1, 2, 3]. Unlike traditional single-locking methods, range locks offer a more refined approach to resource access management. By dividing a shared resource into smaller segments, range locks allow multiple writers to modify different segments simultaneously. This approach overcomes the limitations and bottlenecks of single-lock approaches and fosters parallelism among writers. Consequently, range locks enhance overall efficiency within these critical computing systems.

2 Motivation

Recently, there has been an increase in interest in range-locking techniques. One notable example is that the Linux kernel community is considering using range-locking techniques to replace the `mmap_lock` [1, 4, 5]. The `mmap_lock` uses a per-process semaphore to control access to the whole `mm_struct` [6] and serialize changes to address spaces. Despite previous efforts to overcome the scalability issues of `mmap_lock`, a resolution is yet to be found [5].

In the context of database management systems, range locks also offer a solution to the issue of coarse-grained locking in large databases and indexes. With the storage sizes and the number of pages and indexes increasing exponentially, there are better options than locking the entire index, considering that such a coarse-grained locking mechanism inherently blocks other transactions from progressing, leading to poor throughput and high latency. As a solution to this problem, range locks focus on key ranges, thus allowing multiple transactions to operate concurrently on separate key ranges, reducing memory overhead and lock acquisition bottlenecks [2].

3 Related Work

Previous research has explored various approaches to scalable range locks [7, 8, 9]. The current implementation of range lock in the Linux kernel uses a range tree that keeps track of acquired ranges and an internal spin lock to protect it[7]. Since every range request relies on this single spinlock, it becomes a point of contention.

Song et al. try to improve the Linux kernel's implementation by combining a skip list with a spinlock to manage locked ranges[8]. This technique leverages the skip list, which is more lightweight and efficient than the interval tree and can still conduct intensive searches for overlapping ranges. Despite these advancements, the problem of contention points still requires resolution.

In another research, Kogan et al. designed a range lock based on a linked list, where each node represents an acquired range [9]. By using a linked list, this approach can maintain a lock-free range lock, thus solving the bottleneck problem. However, insertion and lookup operations on the linked list are less efficient than tree-like structures.

4 Approach

In this research's scope, we propose a new concurrent range lock design that leverages a probabilistic concurrent skip list[10, 11]. It consists of two main functions:

- **try_lock**: The `try_lock` function searches for the required range (`[start, start+len]`) in the skip list. If an overlapping range exists, indicating another thread is modifying that range, the requesting thread must wait and retry. If not, the range is added to the list, signaling that the range is reserved.
- **release_lock**: The `release_lock` function releases the lock by finding the address range in the skip list and removing it accordingly.

Our range lock design also utilizes the per-node lock instead of an interval lock, thus addressing the bottleneck problem of the spinlock-based range lock and maintaining the lock's high level of performance.

5 Evaluation

The proposed approach will be evaluated under these evaluation criteria:

- **Performance:** We will test the range lock mechanism under increasing load and concurrent accesses to measure its performance.
- **Correctness:** We will ensure the consistency and correctness of data accesses, especially when there are overlapping data ranges and concurrent operations.
- **Comparison:** We will compare the performance of the proposed solution with existing state-of-the-art approaches.

6 Expected Outcome

We aim to develop a scalable range lock that performs better than the existing range locks. The evaluation results will provide insights into the performance characteristics and potential trade-offs of the proposed mechanism.

7 Resources

We will need 32 cores and 32 GB of RAM for one month. These resources allow us to perform thorough tests under heavy contention and multithreaded scenarios.

Bibliography

[1] J. Corbet. "Range reader/writer locks for the kernel". In: *LWN.net* (2022). Accessed: 2024-04-21. URL: https://lwn.net/Articles/724502/.

[2] Graefe and Goetz. *Hierarchical Locking in B-Tree Indexes*. Accessed: 2024-04-21. 2020. URL: https://dl.gi.de/server/api/core/bitstreams/9f283897-1a24-4714-aa57-c773e6c2da96/content.

[3] C.-G. Lee, S. Noh, H. Kang, S. Hwang, and Y. Kim. *Concurrent File Metadata Structure Using Readers-Writer Lock*. Accessed: 2024-04-21. 2021. URL: https://discos.sogang.ac.kr/file/2021/intl_conf/SAC_2021_CGLee.pdf.

[4] M. Rybczynska. "Introducing maple trees". In: *LWN.net* (2022). Accessed: 2024-04-21. URL: https://lwn.net/Articles/845507/.

[5] J. Corbet. "The ongoing search for mmap_lock scalability". In: *LWN.net* (2022). Accessed: 2024-04-21. URL: https://lwn.net/Articles/893906/.

[6] S. Boutnaru. "Linux Kernel — mm_struct". In: *medium.com* (2023). Accessed: 2024-04-21. URL: https://medium.com/@boutnaru/linux-kernel-mm-struct-fafe50b57837.

[7] J. Kara. "Implement range locks". In: *lkml.org* (2013). Accessed: 2024-04-21. URL: https://lkml.org/lkml/2013.

[8] X. Song, J. Shi, R. Liu, Y. Jian, and H. Chen. *Parallelizing Live Migration of Virtual Machines*. Accessed: 2024-04-21. 2013. URL: https://ipads.se.sjtu.edu.cn/_media/publications:parallel-migration.pdf.

[9] A. Kogan, D. Dice, and S. Issa. *Scalable range locks for scalable address spaces and beyond*. Accessed: 2024-04-21. 2020. URL: https://arxiv.org/pdf/2006.12144.pdf.

[10] H. Maurice, L. Yossi, L. Victor, and S. Nir. "A provably correct scalable concurrent skip list". In: *Conference On Principles of Distributed Systems (OPODIS). Citeseer*. Vol. 103. 2006.

[11] H. Maurice, S. Nir, L. Victor, and S. Michael. *The art of multiprocessor programming*. Newnes, 2020.